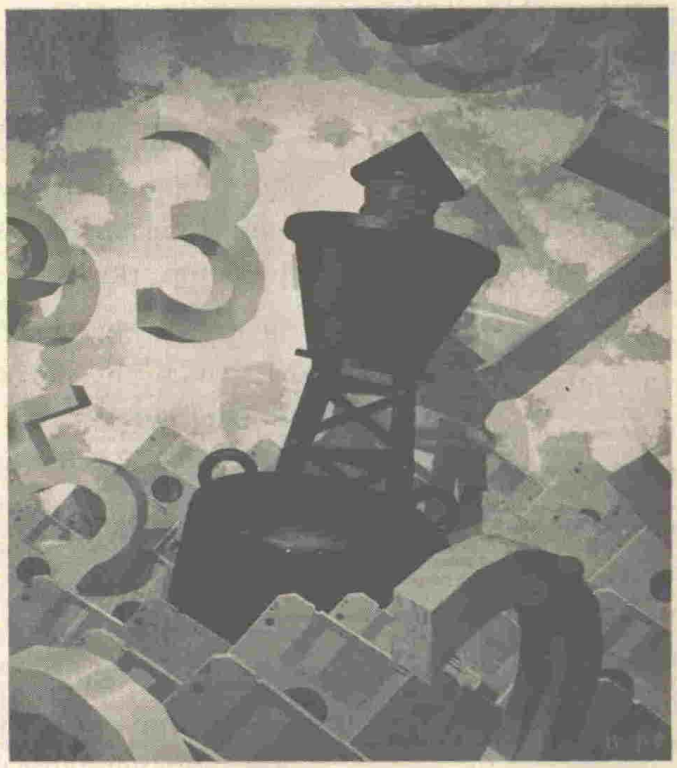
เทคนิคน่ารู้การเขียนโปรแกรม ภาษาปาสคาล ตอนที่ 4

รู้จักวิธีการสร้างระบบไฟล์แบบ "Indexed" เพื่อการจัดเก็บข้อมูลอย่างเป็นระเบียบ ที่ง่าย สะดวก และประหยัดเวลาในการเรียกใช้งาน



ทอร์โบปาสคาลนี้โดยปกติแล้วจะเหมาะกับงานที่วิ่งบนเครื่องพีซีหรือพีซีแลนซึ่งในสมัยแรก ๆ นั้นผู้ผลิตพีซีและพีซีแลนเขาไม่มีซิสเต็มส์ซอฟต์แวร์ด้านจัดไฟล์ (Access Method) ดังนั้นผู้เขียนโปรแกรมจึงอาจต้องเขียนโปรแกรมเพื่อจัดฐานข้อมูลเอง เรื่องนี้นับเป็นอุปสรรคต่อการพัฒนาเป็นอย่างมาก แต่อย่างไรก็ตาม ถึงแม้ในปัจจุบันเราจะมีระบบฐานข้อมูลแล้ว แต่ราคาของซอฟต์แวร์นี้ก็ยังสูง แถมยังบังคับให้เราต้องเพิ่มหน่วยความจำและเพิ่มฮาร์ดดิสก์เข้าไป มันจึงเป็นการเพิ่มค่าใช้จ่ายไปอีกกว่าเท่าตัว ดังนั้นถ้าต้องการประหยัดและไม่ต้องการได้ระบบที่หรูหราอะไรนัก เราก็ควรเรียนรู้วิธีสร้างฐานข้อมูลเอง วันนี้ผมอยากจะพูดถึงวิธีสร้างระบบไฟล์แบบ "Indexed" ในโอกาสหน้าอาจจะพูดถึงระบบ "Indexed Sequential" อีกต่อหนึ่ง

ลักษณะงานของอินเด็กซ์

งานที่เข้าลักษณะ "Indexed" นั้นมีมากมาย เช่น บัญชีกระแสรายวัน ทะเบียนประวัติการพนักงาน ทะเบียนตั๋วเงิน ทะเบียนสินค้าคงคลัง ทะเบียนทรัพย์สิน รายชื่อลูกค้าที่ติดต่ออยู่เสมอ ๆ ทะเบียนเช็ค ลักษณะที่เหมือนกันก็คือเราเก็บเรคอร์ดเอาไว้จำนวนหนึ่ง แล้วเราต้องการเรียกดูหรือเรียกขึ้นมาอัพเดตครั้งละ 1 เรคอร์ด ทั้งนี้โดยไม่มีการจัดเรียงลำดับของการเรียกใช้ดังกล่าว ตัวคีย์ที่จะใช้ในการเรียกหาก็คือ "หมายเลขประจำตัว" หรือ "ชื่อ" หรือ "หมายเลขที่เรากำหนดให้ในตอนเริ่มสร้างเรคอร์ด" ตัวคีย์เพื่อการเข้าถึงดังกล่าวนี้ ปกติจะไม่มีการออกให้ซ้ำกัน การเรียกหาจึงมุ่งเจาะจงไปสู่เรคอร์ดใดเรคอร์ดหนึ่งเท่านั้น แต่เนื่องจากคีย์เพื่อการเรียกหาข้อมูลนี้ไม่ใช่ตัวเลขและไม่เรียงลำดับกัน เราจึงต้องนำมาทำเป็นตารางดัชนี ปัญหาที่เราต้องคิด

และต้องระวังได้แก่ การทำสำเนาข้อมูล (Back up) การตรวจเช็คดูความสอดคล้องของข้อมูล (Integrity) และการกู้ข้อมูล (Recovery) แต่อย่างไรก็ตามเราจะต้องคำนึงถึงความรวดเร็วในการทำงานเป็นประการสำคัญ ทั้งนี้เพราะปัญหาใหญ่ของเครื่องพีซีและพีซีแลนนั้นอยู่ที่เป็นเครื่องขนาดเล็ก มีความสามารถต่ำ มีจานแม่เหล็กที่ไม่เร็วนัก และในบางครั้งเราอาจใช้ดิสเกตต์เป็นตัวเก็บข้อมูลซึ่งทำงานได้ช้ากว่าฮาร์ดดิสก์หลายเท่าตัว

ข้อกำหนดเพื่อการออกแบบ

ด้านฟังก์ชันปกติ จะต้องมีความสามารถในการโหลด (Load) และอันโหลด (Unload) ฐานข้อมูลได้ จะต้องอัพเดตแบบ Add Record, Change Record และ Delete Record ได้

ด้านฟังก์ชันตรวจเช็ค จะต้องมีความสามารถตรวจหาความผิดปกติในรูปแบบต่าง ๆ ได้ เช่น มีเรคอร์ดแต่ไม่มีอินเด็กซ์ มีอินเด็กซ์แต่ไม่มีเรคอร์ด มีคีย์ซ้ำกัน รหัสสถานะเรคอร์ดผิดปกติ...ฯลฯ

ด้านการแก้ปัญหา จะต้องมีความสามารถที่จะทำสำเนาข้อมูลและนำกลับมาใช้ได้โดยง่าย ต้องมีระบบเตือนเมื่อไฟล์ใกล้จะเต็ม และจะต้องไม่ทำให้เรคอร์ดขาดหายเมื่อเกิดไฟฟ้าดับหรือโปรแกรมเกิด Abnormal End Of Job

ด้านประสิทธิภาพ ต้องทำงานได้เร็ว เข้าใจง่าย ขยายขนาดของไฟล์ได้โดยง่าย

ครับ การตั้งข้อกำหนดนั้นมันง่าย แต่การออกแบบให้ได้ครบถ้วนทุกข้อนั้นยาก โดยมากแล้วข้อกำหนดมันขัดแย้งกันเอง เช่น ถ้าต้องการประสิทธิภาพก็อาจด้อยทางด้านความปลอดภัย สำหรับระบบ Indexed File ที่ผมจะพูดถึงนี้จะเน้นทางด้านประสิทธิภาพแต่ก็ทำข้อกำหนดอื่น ๆ ได้ดีพอสมควร ผมจะขอเริ่มจากด้านฟังก์ชันปกติเสียก่อน แล้วเติมวิธีการที่จะให้ได้ข้อกำหนดอื่น ๆ เข้าไป

ทบทวนความจำ

ก่อนอื่นผมต้องขอรื้อฟื้นความจำในเรื่องไฟล์ของภาษาปาสคาลเสียก่อน ไฟล์ในปาสคาลมีลักษณะก้ำกึ่งระหว่างไฟล์แบบสุ่ม (Random) และไฟล์แบบเรียงลำดับ (Sequential) กล่าวคือเราสามารถจะเข้าถึงเรคอร์ดที่เท่าใดโดยทันทีเลยก็ได้ทั้งนี้โดยใช้คำสั่ง SEEK (File name, Relative-Key) เจ้า Relative-Key นี้คือตัวบอกตำแหน่งของเรคอร์ดที่เราต้องการ หลังจากสั่งให้ SEEK แล้วก็สามารถสั่ง READ หรือ WRITE ได้ เครื่องก็จะกระทำต่อเรคอร์ดที่สั่งให้ SEEK ไว้ แต่ท่านควรสังเกตว่า คำสั่ง READ หรือ WRITE นี้จะไม่มีการบอกตำแหน่งที่ของเรคอร์ด มันกลับเขียนว่า READ (File name, Work-Area) และ WRITE (File name, Work-Area) และเราสามารถสั่ง READ หรือ WRITE ต่อไปอีกได้โดยไม่ต้องสั่ง SEEK ในกรณีเช่นนี้มันจะกระทำกับเรคอร์ดที่อยู่ถัดไปตามลำดับ หรือคือทำงานแบบเรียงลำดับต่อไปได้เลย

นอกจากลักษณะการเข้าถึงเรคอร์ดแบบแปลกประหลาดแล้วเจ้าตัวชี้เรคอร์ดหรือ Relative-Key นั้น ในภาษานี้ยังเริ่มต้นที่เลข 0 (ศูนย์) ด้วย ซึ่งมักจะทำให้เข้าใจยากเพราะการที่เราจะเรียกเรคอร์ดแรกว่า "เรคอร์ดที่ศูนย์" นั้นมันฟังดูพิลึก ส่วนเรคอร์ดที่ 2 จะกลายเป็น "เรคอร์ดหมายเลข=1"...ฯลฯ เพื่อลดปัญหาความสับสนในเรื่องนี้ผมแนะนำให้ใช้เรคอร์ดที่ 2 เป็นต้นไปเพื่อการเก็บข้อมูล ซึ่งคีย์ของมันจะเป็นหมายเลข 1, 2, 3,...ตามลำดับ ส่วนเรคอร์ดแรกหรือเรคอร์ดที่ "0" นั้นผมจะนำไปใช้ประโยชน์อย่างอื่น ซึ่งจะอธิบายกันต่อไป

อธิบายศัพท์

มาถึงตรงนี้ผมคงต้องขอเคลียร์คำศัพท์สัก 2-3 คำ เพราะถ้าไม่รีบเคลียร์กันเสียก่อนก็จะทำให้สับสนได้ง่าย

Symbolic-Key

นี่คือ "ชื่อ" หรือ "หมายเลขประจำตัว" หรือ "หมายเลขบัญชี" หรืออะไร ๆ ที่เราใช้ในการบอกว่าเราต้องการข้อมูลของคนใดหรือบัญชีใดออกมาเพื่ออัพเดต ซิมโบลิคคีย์นี้จะเป็นตัวเลขหรือตัวอักษรหรือทั้งตัวเลขและตัวอักษรปนกันก็ได้ โดยทั่ว ๆ ไปเราจะกำหนดว่าต้องเป็นสตริง และมีความยาวเท่ากันทุกครั้ง ซิมโบลิคคีย์นี้จะมีค่าซ้ำกันไม่ได้ และค่าของมันก็ไม่จำเป็นต้องมีเรียงกันไปอย่างครบถ้วน การเปรียบเทียบค่าของ ซิมโบลิคคีย์ (ใน IF Statement) มันจะให้ผลเป็นมากกว่า เท่ากับ หรือน้อยกว่า โดยดูจากการจัดลำดับตัวอักษรในตารางแอสกี (เรียกว่า Collating Sequence) เรื่องนี้เปิดดูตารางท้ายคู่มือเทอร์โบปาสคาลได้ เช่น อักษร A มีค่าน้อยกว่า B, C, D,... แต่มีค่าสูงกว่าเลข 0, 1, 2, 3,... ส่วนตัวอักขระไทยนั้นจะมีค่าสูงกว่าพวกภาษาอังกฤษเหล่านี้

Relative-Key

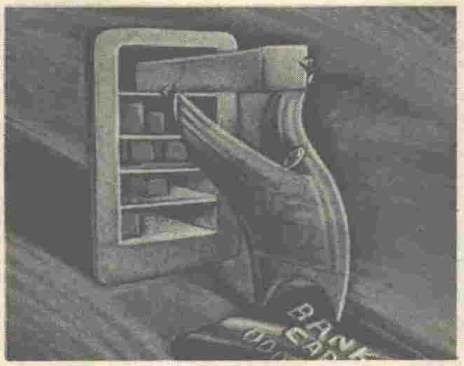
นี่คือตัวเลขประจำเรคอร์ด ซึ่งผมได้พูดไว้แล้วว่าในภาษาปาสคาลนี้จะมีค่าได้ตั้งแต่ 0 เป็นต้นไป ในภาษาปาสคาลเวอร์ชัน 3.0 จะต้องเป็นประเภท Integer ระบบไฟล์ของเวอร์ชันนี้จึงมีได้ไม่เกิน 32676 เรคอร์ด เจ้ารีเลตีฟคีย์นี้ จะใช้ในคำสั่ง SEEK เพื่อให้เราเข้าถึงเรคอร์ดที่เท่าใดก็ได้โดยทันที ในการค้นหาข้อมูลนี้ในบางครั้งเราต้องมีการเปรียบเทียบค่าของรีเลตีฟคีย์ ซึ่งจะให้ผลเป็น มากกว่า เท่ากับ หรือน้อยกว่า ทั้งนี้โดยดูค่าที่เป็น Integer ดังกล่าว

File Pointer

เจ้าตัวนี้จะเป็นตัวเดียวกับคำว่า "Relative-Key" เขาใช้ในการอธิบายคำสั่ง SEEK และ RESET คำสั่ง SEEK เป็นการเซตค่าไฟล์พอยน์เตอร์ไปยังตำแหน่งที่ต้องการอ่านหรือบันทึก ส่วน RESET เป็นการเซตค่าไฟล์พอยน์เตอร์ไปที่เลข 0 (ศูนย์) ในบทความของผม ผมจะพยายามใช้แต่คำว่ารีเลตีฟคีย์ เพราะมันเป็นคำชัดเจนมากกว่า

End-of-File Pointer

นี่คือ ตัวเลขที่ดอสจดเอาไว้ในเฮดเดอร์ของไฟล์เพื่อบอกว่าไฟล์นี้ในขณะนี้ได้เคยใช้คำสั่ง WRITE ไปที่เรคอร์ดที่เท่าใดแล้วมันเป็นค่ารีเลตีฟคีย์ที่สูงที่สุดที่เคยถูกใช้มา ถ้าเราไปสั่ง READ ณ จุดที่มีรีเลตีฟคีย์สูงกว่านี้ มันจะเกิด "Error" หรือคือทำให้โปรแกรมเราแฮงได้แต่ถ้าเราสั่ง WRITE (หรือ SEEK แล้ว WRITE) ที่จุดที่มีรีเลตีฟคีย์สูงกว่า End-of-File Pointer นี้ เครื่องจะหาเนื้อที่ในดิสค์เพิ่มให้ได้แล้วจะบันทึกเรคอร์ดนั้นลงไว้ และปรับ



ค่า End-of-File Pointer เสียใหม่ แต่อย่างไรก็ตาม ค่า End-of-File Pointer ตัวใหม่นี้จะยังไม่ถูกบันทึกลงในจานแม่เหล็ก มันจะค้างอยู่ในหน่วยความจำของดอส เรื่องนี้อาจทำให้เรคอร์ดใหม่ ๆ หาไม่พบได้เมื่อเกิดไฟดับ หรือเมื่อเราไปหยุดโปรแกรมโดยยังไม่ทันสั่งให้ปิดไฟล์ (Close File)

อนึ่งท่านผู้อ่านควรสังเกตว่าไฟล์ในภาษาปาสคาลนี้ มันเปิดขึ้นมาแล้วใช้เป็นทั้งอินพุตและเอาต์พุตพร้อมกัน แต่เรคอร์ดที่เราสั่ง WRITE เอาไว้นั้นมันอาจมีครบตั้งแต่เลขที่ 0 ไปจนถึง End of File Pointer ก็ได้ หรือจะมีไม่ครบก็ได้ ส่วนใหญ่แล้วเรามักจะ WRITE ไว้ไม่ครบ เช่น เราอาจ WRITE ไว้ที่ 0, 1, 2, 7, 12, 25, 39, 80, 850 ก็เป็นได้ ในกรณีเช่นนี้ End-of-File Pointer จะชี้ไปที่ 850 อย่างนี้ก็จะมีคำถามว่าเรคอร์ด 3, 4, 5, 6, 8, 9, 10, 11,...849 มันอยู่ที่ไหน และเราจะ READ หรือ WRITE ได้หรือไม่

เรื่องนี้เข้าใจยากหน่อยคือ ปาสคาลจะจัดหาเนื้อที่จากจานแม่เหล็กให้เราทีละบล็อกเล็ก ๆ ซึ่งอาจเพียงพอสำหรับบันทึกเรคอร์ดให้เราจำนวนหนึ่ง เพื่อความสะดวก ผมจะสมมติว่าจัดให้บล็อกละ 10 เรคอร์ด อย่างนี้เราจะพบว่าเรคอร์ด 3, 4, 5, 6, 8, 9 นั้นมีอยู่แล้ว แต่มันยังเป็นขยะ (Garbage) อยู่ ท่านสามารถสั่ง READ ได้ เครื่องจะส่งขยะมาให้ ท่านสามารถสั่ง WRITE ได้ เครื่องจะไม่ไปเรียกหาเนื้อที่จานแม่เหล็กเพิ่มเติม

ทั้งหมดนี้เป็นเพราะการมีเรคอร์ด 0, 1, 2 ไว้แล้วนั้นมันต้องมีเนื้อที่ดิสค์สำหรับ 1 บล็อกให้เราแต่การที่เรามีเรคอร์ดที่ 80 แล้วกระโดดไปถึง 850 นั้นเราจะพบว่าเรคอร์ดที่ 90-99, 100-109,...840-849 จะยังไม่มีอยู่ในจานแม่เหล็ก ถ้าท่านสั่ง READ มันจะส่งขยะจากอวกาศมาให้ (คือ คาดเดาไม่ได้ มันมีค่าไม่แน่ไม่นอน) ถ้าท่านสั่ง WRITE มันจะไปขอเนื้อที่ในจานแม่เหล็กมาเพิ่มให้ ที่จุดนี้ท่านอาจเกิดปัญหาได้ คือ ถ้าจานแม่เหล็กเราเต็มมันจะทำให้โปรแกรมแฮงได้ แต่ถ้าหาเนื้อที่เพิ่มได้ท่านก็จะทำงานต่อได้ ถ้าท่านไปดับไฟหรือจบโปรแกรมก่อนสั่งปิดไฟล์ ท่านก็อาจมีปัญหาได้อีก คือ ตัวเรคอร์ดจริง ๆ นั้นมีเกิดขึ้นในจานแม่เหล็กแต่ดอสอาจยังไม่ได้อัพเดต FAT (File Allocation Table) ในจานแม่เหล็กเมื่อเปิดเครื่องใช้ใหม่ก็อาจหาเรคอร์ดใหม่เหล่านี้ไม่พบ

FAT คือตัวที่ดอสใช้สำหรับชี้ไปว่าไฟล์ของเราถูกกระจายไปอยู่ตรงไหนในดิสค์บ้าง ดอสรุ่นเก่า ๆ นั้นต้องการประหยัดเวลา มันจึงไม่อัพเดต FAT ในดิสค์ทุกครั้งที่มีการแก้ไข แต่ดอสรุ่นใหม่ ๆ นั้นจะมีการเก็บภาพเมมโมรี (รวมทั้ง FAT ในเมมโมรี) เอาไว้เป็นระยะ ๆ มันจึงอาจปรับ FAT ได้ถูกต้องหลังไฟฟ้าดับได้ แต่เรื่องนี้ท่านอย่าเสี่ยงดีกว่า หาหนทางป้องกันข้อมูลหายด้วยตนเองจะดีกว่า วิธีที่ง่ายที่สุดคือสั่งปิดไฟล์ทุกครั้งหลังอัพเดตเสร็จไป

1 รายการ วิธีนี้จะบังคับให้ดอสมันอัพเดต End of File Pointer และ FAT ในจานแม่เหล็กให้เรา

ครับ คงรู้จักความหมายของคำศัพท์ที่อธิบายกันไปแล้วและผมยังมีคำศัพท์ที่สำคัญอีก 3-4 คำ เช่น Index, Hashing, Index-Entry, Link หรือ Pointer ที่จะต้องอธิบาย แต่ผมจะขอเก็บไว้พูดในตอนที่เนื้อเรื่องกล่าวไปถึงจะดีกว่า มาถึงตรงนี้ท่านก็คงพอจะเข้าใจลักษณะการทำงานของไฟล์ธรรมดา ๆ ในภาษาปาสคาลแล้วล่ะ ในขั้นต่อไปผมก็จะพูดถึง เรื่องของการสร้างแฟ้มข้อมูลหลัก (Master File) และแฟ้มดัชนี (Index File) กัน

แฟ้มข้อมูลหลัก

คือแฟ้มที่ใช้เก็บข้อมูล เราเก็บตามลำดับที่เกิดขึ้นของข้อมูล แฟ้มนี้เราควรสร้างขึ้นด้วยการเคลียร์ให้เป็นเรคอร์ดเปล่า ๆ เอาไว้เป็นจำนวนที่เราคาดว่าจะต้องใช้ใน 3-5 ปี สำหรับเรคอร์ดที่ 0 นั้นเราจะใช้เป็นตัวชี้ว่าเราได้บรรจุข้อมูลเข้าไปจนถึงเรคอร์ดที่เท่าใดแล้ว (ตัวเลขที่เราเก็บจริง ๆ นั้นจะเป็นรีเลตีฟคีย์ของเรคอร์ดเปล่าอันถัดไป หรือเรียกตามภาษาฝรั่งว่า "Next Available Record")

การสร้างแฟ้มข้อมูลหลักที่เป็นเนื้อที่เปล่า ๆ นี้ ควรทำด้วยโปรแกรมเล็ก ๆ อันหนึ่ง ซึ่งเขียนได้ง่ายมาก มันจะเริ่มต้นด้วยการสั่ง REWRITE (File name) คำสั่งนี้จะยกเลิกไฟล์เก่าที่ชื่อเดียวกันนี้ เสร็จแล้วให้ SEEK เรคอร์ดที่ 0 แล้วบันทึกไว้ว่า Next Available Record มีค่าเท่ากับ 0001 ต่อจากนั้นก็วนลูปเพื่อสั่ง WRITE (File name, Work-Area) ไปเท่ากับ

จำนวนเรคอร์ดเปล่าที่ต้องการ เจ้า Work-Area นี้เราจะต้องสร้างให้มีรูปร่างเป็นเรคอร์ดที่ไม่มีข้อมูล เช่น ฟิลด์ที่เป็นตัวเลขก็เซตให้เป็น 0 เมื่อวนจบแล้วก็สั่งปิดไฟล์และจบโปรแกรม ด้วยวิธีนี้เราจะได้แฟ้มข้อมูลที่มีเนื้อที่จริงไว้รองรับทุกเรคอร์ด ปัญหาเรื่องโปรแกรมจะแฮงเมื่อขอเนื้อที่แม่เหล็กไม่ได้ก็จะไม่เกิดขึ้น และปัญหาเรื่องไฟฟ้าดับแล้วข้อมูลใหม่หายเพราะ End-of-File Pointer ยังไม่ถูกอัพเดตก็จะไม่เกิดขึ้น ทั้งนี้เพราะเราจะไม่มีการใช้เนื้อที่เกินกว่าที่เรากำหนดไว้ในตอนแรกนี้ เจ้า End-of-File Pointer จะชี้ไปที่เรคอร์ดสุดท้ายตลอดเวลา

อนึ่ง ถ้าท่านต้องการให้มีประสิทธิภาพยิ่งขึ้นท่านอาจต้องสั่งให้ดอสทำการจัดเนื้อที่ในดิสก์ให้แน่น (Condense Disk) เสียก่อนที่จะวิ่งโปรแกรมนี้ ถ้าท่านทำเช่นนี้ได้ท่านก็จะได้แฟ้มข้อมูลหลักที่เป็นเนื้อที่ผืนเดียวต่อเนื่องกัน

แฟ้มดัชนี

คือแฟ้มที่จะเก็บซิมโบลิคคีย์กับตำแหน่งที่อยู่ของเรคอร์ดเป็นคู่ ๆ เอาไว้ มันคือ ตารางที่จะบอกว่าเรคอร์ดไหนอยู่ที่ตรงไหนในแฟ้มข้อมูลหลัก ยกตัวอย่างเช่น ถ้าเราเอาข้อมูลของบัญชีหมายเลข 12345 ไปบันทึกลงไว้ที่เรคอร์ด 0001 ของแฟ้มข้อมูลหลัก เราก็จะจับเอาสตริง "12345" มาถ้าคู่กับตัวเลข 0001 แล้วนำไปเก็บไว้ที่ใดที่หนึ่งในแฟ้มดัชนีนี้ ท่านผู้อ่านควรสังเกตว่าเลข 0001 คือ รีเลตีฟคีย์ ส่วน "12345" คือ ซิมโบลิคคีย์ข้อมูล 2 ชิ้นนี้เมื่อนำมาวางชิดติดกันเราเรียกว่า "Index-Entry" คำนี้บางคนเรียกมันย่อ ๆ ว่า "Index" และบางคน (ในตำรา) เรียกมันย่อ ๆ ว่า "Entry" ซึ่งในภาษาไทยของเราเรียกว่า "ดัชนี" เพียงอย่างเดียว

จำนวนดัชนีหรือ "เอ็นทรี้" นี้จะต้องมีเท่ากับจำนวนเรคอร์ดที่มีอยู่ในแฟ้มข้อมูลหลัก แต่ในทางปฏิบัติเรามักจะต้องเตรียมเนื้อที่ไว้รองรับดัชนีมากกว่าจำนวนเรคอร์ดที่เตรียมไว้ในแฟ้มข้อมูลหลัก ทั้งนี้เพราะจะมีเนื้อที่จำนวนหนึ่งในแฟ้มดัชนีที่ต้องว่างทิ้งไว้ เรื่องนี้จะเข้าใจก็เมื่อรู้วิธีการจัดเก็บดัชนีเสียก่อน

วิธีการจัดเก็บดัชนี

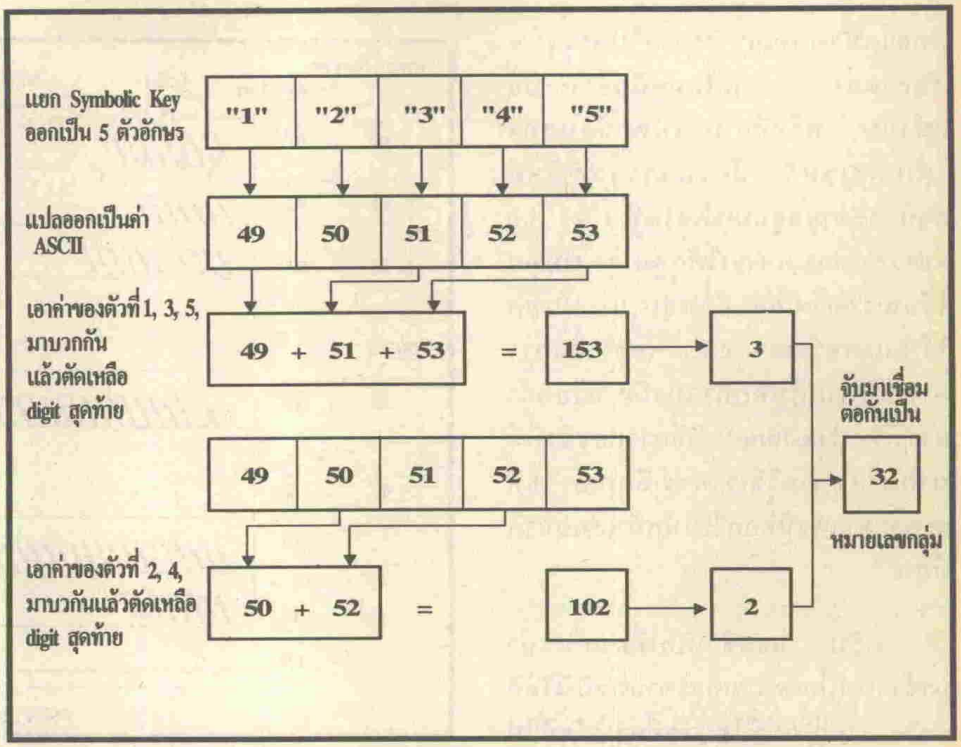
วิธีการจัดเก็บดัชนีหรือเอ็นทรี้นั้นทำได้มากมายหลายวิธี วิธีที่ผมจะขอแนะนำในบทความนี้คือให้แบ่งเป็นกลุ่มย่อย ๆ n กลุ่ม ค่า n นี้จะขึ้นกับขนาดของแฟ้มข้อมูลหลัก วิธีประมาณการทำได้ดังนี้ คือ เราจะเริ่มจากการถามตัวเองเสียก่อนว่าเราจะจัดเก็บดัชนีไว้กี่ตัวต่อ 1 เรคอร์ดของแฟ้มดัชนี แฟ้มดัชนีควรใช้เรคอร์ดที่มีขนาดเป็นตัวคูณของ 256 ไบต์ หรือคือมีขนาด 256 หรือ 512 หรือ 1024 หรือ 2048 ไบต์ ทั้งนี้เพื่อให้มันสอดคล้องกับขนาดของฟิสิคัลเซกเมนต์ของจานแม่เหล็ก ในงานขนาดเล็กเรามักจะเลือกขนาดจานเล็กตามไปด้วย สมมติว่าแฟ้มหลักมีขนาด = 4800 เราอาจเลือกขนาดของอินเด็กซ์เรคอร์ด = 512 ไบต์ จากตัวเลขนี้เรา

ก็มาคำนวณว่าเราจะเก็บดัชนีได้เรคอร์ดละกี่ตัว เรื่องนี้ขึ้นอยู่กับความยาวของซิมโบลิคคีย์และรีเลตีฟคีย์ที่ใช้ ในตัวอย่างนี้ ซิมโบลิคคีย์เป็นสตริงขนาด 5 ไบต์ มันจึงกินเนื้อที่ 6 ไบต์ ส่วนรีเลตีฟคีย์นั้นเป็นเลขจำนวนเต็มจึงกินเนื้อที่ 2 ไบต์ รวมกันแล้วจึงเป็นเนื้อที่ 8 ไบต์

ดังนั้น 1 เรคอร์ดของแฟ้มดัชนีจะบรรจุได้ 64 ดัชนี แต่เราอาจต้องใช้เนื้อที่บางส่วนเพื่อการชี้โยงไปยังเรคอร์ด overflow ดังนั้นเราอาจเลือกว่าจะบรรจุดัชนีไว้ 60 ดัชนีต่อ 1 เรคอร์ด จากตัวเลขอันนี้เราก็ประมาณได้ว่าแฟ้มดัชนีต้องมี 4800/60=80 เรคอร์ด เป็นอย่างน้อย

จากตัวเลขอันนี้เราก็อาจเลือกว่าจะให้ n=100 และจำนวนเรคอร์ดจริง ๆ ของแฟ้มดัชนีควรมี 120 เรคอร์ด ทั้งนี้โดยให้เรคอร์ดที่ 1 ถึง 100 จะเป็นของกลุ่มที่ 00 ถึง 99 ส่วนเรคอร์ดที่ 101 ถึง 120 นั้นเผื่อเอาไว้เพื่อใช้กับกลุ่มที่มีดัชนีมาเก็บจนล้นเรคอร์ด อนึ่งขอให้สังเกตว่าค่า n นั้นควรจะเป็นตัวคูณของ 10 หรือคือเป็น "10" หรือ "100" หรือ "1,000" หรือ "10,000" ทั้งนี้เพื่อความสะดวกในการคำนวณหากลุ่มจากซิมโบลิคคีย์ ซึ่งจะได้กล่าวต่อไป แต่เรื่องนี้ก็มิใช่กฎที่ตายตัว เราจะจัดกลุ่มเป็นกี่กลุ่มก็ได้ทั้งนั้น

ครับ หลังจากวางแผนกว้าง ๆ เอาไว้ตามที่กล่าวมาข้างบนเราก็ต้องมาสร้างแฟ้มดัชนีเปล่า ๆ ขึ้นมาไว้ ลักษณะของเรคอร์ดจะประกอบไปด้วยที่ว่างสำหรับเก็บดัชนี 60 ตัวส่วนที่เหลืออีก 32 ไบต์นั้นเราจะต้องใช้เพื่อชี้โยงไปยังเรคอร์ดที่ 101 หรือ 102...120 ตัวชี้โยงนี้เราเรียกว่า "Link" หรือ "Next Index Record Pointer" มันเป็นตัวเลขชนิดจำนวนเต็ม (เพราะเป็น Relative Key ภายในแฟ้มดัชนีเอง) จึงกินเนื้อที่ 2 ไบต์ ในตอนต้นเราจะต้องเซตค่านี้ให้เป็น 0 (ศูนย์) ทั้งนี้เพื่อบอกว่าไม่มีการเชื่อมโยงไปที่ไหน



รูปที่ 1 วิธีคำนวณหา "กลุ่ม" จาก ซิมโบลิคคีย์

การสร้างแฟ้มดัชนีนี้ทำได้คล้าย ๆ กับการสร้างแฟ้มข้อมูลหลัก คือเขียนเป็นโปรแกรมเล็ก ๆ เพื่อเคลียร์พื้นที่เอาไว้ 120 เรคอร์ดแล้วก็สั่งปิดไฟล์แล้วก็หยุดไปเลย

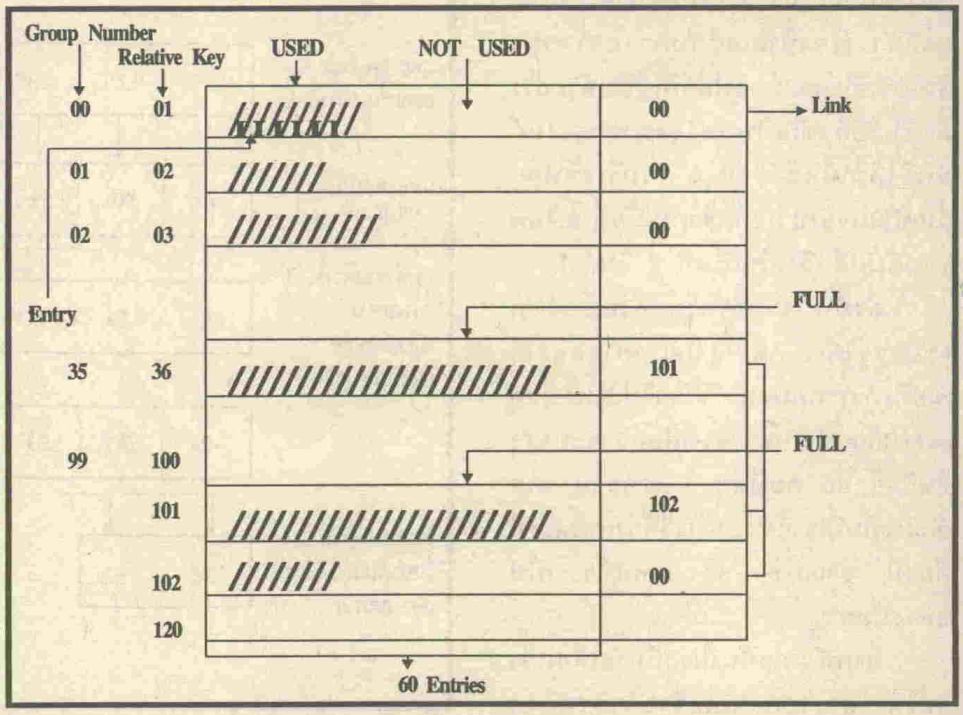
วิธีการจัดกลุ่มดัชนี

ทีนี้ก็จะมาถึงวิธีการจัดกลุ่ม นี่เป็นหัวใจของการจัดแฟ้มแบบ Index คำว่าดัชนีหรืออินเด็กซ์นี้จริง ๆ แล้วก็มีความหมายถึง การจัดกลุ่ม เช่นบรรณารักษ์นั้นเมื่อได้หนังสือหรือตำราเข้ามาแล้วก็จะให้หมายเลขเพื่อจัดกลุ่มหนังสือ ซึ่งเขาใช้คำว่าอินเด็กซ์เช่นกัน การจัดกลุ่มจะช่วยให้เราค้นหาได้เร็วขึ้น ในระบบคอมพิวเตอร์นั้นมีวิธีการจัดกลุ่มอยู่มากมายแต่ผมจะขอแนะนำวิธีง่าย ๆ อันหนึ่ง คือ เมื่อ n ของเรามีค่าเท่ากับ 100 มันก็จะเป็นตัวเลข 2 หลัก ตัวเลขนี้จะต้องมาจากตัวอักษรต่าง ๆ ที่อยู่ในซิมโบลิคคีย์ของเรา ในกรณีนี้ผมแนะนำให้เอาตัวอักษรที่ 1, 3, 5 มาแปลงรหัสเป็นแอสกี แล้วนำมาบวกกัน แล้วตัดให้เหลือแต่หลักหน่วย แล้วใช้เป็นตัวเลขหลักหน้าของตัวเลขบอกกลุ่ม และให้เอาตัวอักษรที่ 2, 4 มาแปลงรหัสเป็นแอสกี แล้วนำมาบวกกันแล้วตัดให้เหลือแต่หลักหน่วย แล้วใช้เป็นเลขหลักหลังของตัวเลขบอกกลุ่ม

รูปที่ 1 แสดงเป็นขั้นตอนของการคำนวณหา "กลุ่ม" ตามที่กล่าวมาข้างบนนี้ ขอให้ท่านผู้อ่านสังเกตว่าวิธีการดังกล่าวนี้จะทำให้ซิมโบลิคคีย์ที่มีค่าต่าง ๆ กันนั้นจะถูกกระจายไปยังกลุ่มต่าง ๆ ในอัตราเฉลี่ยที่ค่อนข้างจะเท่า ๆ กัน ซึ่งจะทำให้ตัวดัชนีของ

เรคอร์ดทั้งหมดถูกกระจายไปบรรจุบนเรคอร์ดต่าง ๆ ในแฟ้มดัชนีอย่างเสมอหน้ากัน หรือคือเมื่อโหลดข้อมูลลงแฟ้มข้อมูลหลักได้ครบแล้วจะทำให้มีดัชนีกระจายอยู่บนเรคอร์ดที่ 1 ถึง 100 อย่างสม่ำเสมอ (เราให้กลุ่ม 00 บันทึกไว้บนเรคอร์ดเลข 1, กลุ่ม 01 บันทึกไว้บนเรคอร์ดเลข 2,... ฯลฯ) ในการคำนวณหากลุ่มที่กล่าวมานี้เขาเรียกว่าการทำ "Hashing" วิธีการแฮชชิ่งนี้มีมากมายหลายวิธีเราควรเลือกเอาวิธีที่จะกระจายดัชนีออกไปให้สม่ำเสมอมากที่สุด

ครับ แฮชชิ่งแบบที่ผมแนะนำมาข้างบนนี้อาจช่วยกระจายดัชนีนี้ได้ดีก็จริง แต่มันก็มีโอกาสที่จะมีดัชนีไปตกในกลุ่มเดียวกันเกิน 60 ตัว โดยเฉพาะอย่างยิ่งเมื่อเรามีการเพิ่มข้อมูลเข้าไปในแฟ้มข้อมูลหลักไปมาก ๆ ในกรณีเช่นนั้นเราจะทำอย่างไร เรื่องนี้ผมให้ไปเอาเรคอร์ดที่ 101, 102, 103,...มาใช้เรคอร์ดที่เผื่อเอาไว้ (spare) เหล่านี้ใครต้องการก่อนก็เรียกใช้ก่อนไปตามลำดับ เราจะใช้เรคอร์ดที่ "0" เป็นตัวเก็บค่าที่จะชี้บอกว่าเรคอร์ดว่างตัวถัดไปอยู่ที่ไหน ค่านี้เราเรียกว่า "Next Available Record" เช่นกัน มันจะมีค่าเท่ากับ 101 ในตอนเริ่มต้น สมมติว่าตอนนี้เรากำลังจะเพิ่มดัชนีเข้าไปในกลุ่มที่ 35 เราได้เรียกเอาเรคอร์ดที่ 36 (ตรงกับกลุ่มที่ 35) ขึ้นมาดูแล้ว แล้วพบว่ามันมีดัชนีอยู่เต็ม 60 แล้ว ดังนั้นเราจึงต้องไปอ่านเรคอร์ด "0" ขึ้นมา เรคอร์ด "0" จะบอกว่าเรคอร์ดที่ว่างอยู่ในอันดับถัดไปอยู่ที่ไหน ค่านี้คือ ตัวที่จะเชื่อมโยงเรคอร์ดเข้าด้วยกัน เราต้องเอาค่านี้ไปใส่ในช่อง "ลิงค์" ที่อยู่ในเรคอร์ดที่ 36 สมมติว่า "ลิงค์" นี้มีค่าเป็น 101 เราก็จะต้องอ่านเอาเรคอร์ดที่ 101 ขึ้นมาแล้วบรรจุดัชนีตัวใหม่ลงไปที่นี่ เสร็จแล้วก็บันทึกกลับลงในจานแม่เหล็ก การเชื่อมโยงโดยใส่รีเลตีฟคีย์ (ของเรคอร์ดที่ต่อออกไป) ไว้ในช่อง "ลิงค์" ของเรคอร์ดต้นทางนี้เราเรียกว่าเป็น "Foreward Link" คือชี้เชื่อมไปข้างหน้าทางเดียว มันไม่มีการชี้โยงย้อนกลับ มันมีข้อดีตรงที่ว่าทำงานง่าย เขียนโปรแกรมเพื่อค้นหาข้อมูลได้ง่าย แต่ถ้าเกิดความชำรุดเสียหายของตัวเชื่อมโยงเราจะหาที่ผิดได้ยาก รูปที่ 2 แสดงตัวอย่างลักษณะของแฟ้มดัชนีเมื่อมีการบรรจุดัชนีลงไปจำนวนหนึ่งแล้ว ในรูปนี้เราสมมติว่ากลุ่ม 35 นั้นได้บรรจุลงไปในเรคอร์ดเริ่มต้นจนเต็มแล้ว มันจึงต้องเชื่อมโยงชี้ไปยังเรคอร์ดที่ 101 และในตัวอย่างนี้ก็ได้บรรจุดัชนีลงในเรคอร์ดที่ 101 จนเต็มอีก เราจึงเชื่อมโยงไปยังเรคอร์ดที่ 102 ให้



รูปที่ 2 ลักษณะการบรรจุดัชนี และการเชื่อมโยงไปข้างหน้า

วางแนวคิดในการโหลดข้อมูล

โอเค เราได้ภาพกว้าง ๆ ของแฟ้มข้อมูลหลักและแฟ้มดัชนีแล้ว ในขั้นต่อไปก็ต้องมาวางแนวความคิดว่าเราจะโหลดข้อมูลได้อย่างไร เราจะอัพเดตแบบ Add-Record ได้อย่างไร เราจะอัพเดตแบบ Change-Record ได้อย่างไร และเราจะอัพเดตแบบ Delete-Record ได้อย่างไร เรื่องแรกที่ต้องคิดก็คือ เมื่อเราสั่ง Delete-Record นั้นเราจะทำอย่างไร

ข้อเสนอแนะของผมก็คือ เราจะยังปล่อยข้อมูลในแฟ้มข้อมูลหลักค้างอยู่ แต่เราจะมีไบต์หนึ่งบอกสถานะของมันว่าถูกลบไปแล้ว เจ้า "สถานะ" นี้ต้องมีไว้ทุกเรคอร์ด สถานะจะมีได้ 3 อย่าง คือ 0=ว่างอยู่ (Empty) 1=ใช้

งาน (Active) และ 2=ยกเลิก (Deleted) ไปแล้ว สำหรับตัวดัชนีของเรคอร์ดนี้เราก็ยังปล่อยค้างเอาไว้ในแฟ้มดัชนี แต่เราจะเปลี่ยนเจ้ารีเลตีฟคีย์ในเอนทรี้นี้ให้เป็นเลข 0 วิธีนี้เป็นวิธีที่ง่าย เราจะลบเรคอร์ดและดัชนีเหล่านี้ทิ้งไปจริง ๆ ก็เมื่อเราไปทำฟังก์ชันโหลดอีกครั้งหนึ่ง

ฟังก์ชันโหลดข้อมูล

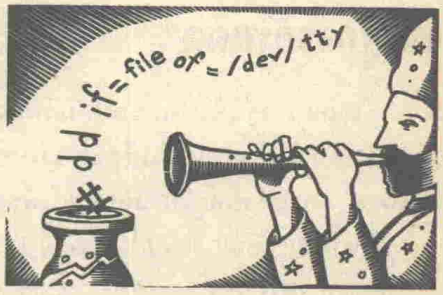
หลังจากวางแนวคิดเรื่องดิลีทแล้ว เราก็มาดูเรื่องการโหลดข้อมูล ข้อมูลที่เราแบคอัพเอาไว้ หรือที่เจาะเตรียมเอาไว้นั้นจะต้องเป็นซีเควนเชียลไฟล์ มันคือ ข้อมูลบัญชีของนาย ก นาย ข นาย ค...ฯลฯ ข้อมูลพวกนี้เราต้องรู้ว่าแบ่งฟิลด์อย่างไร และแต่ละฟิลด์มีลักษณะข้อมูลเป็นอะไรบ้าง ถ้าเราต้องการป้องกันความผิดพลาด เราก็อาจต้องทำการตรวจสอบลักษณะข้อมูล (Data Editing) ไปพร้อม ๆ กับการโหลดลงฐานข้อมูลของเรา แต่ถ้าเราเชื่อมั่นว่าข้อมูลเหล่านี้มาจากระบบที่ตรวจสอบไว้แล้ว เราก็ไม่ต้องตรวจสอบซ้ำ เราสามารถโหลดลงไปได้เลย การโหลดนี้จะกระทำทีละเรคอร์ด

ขั้นตอนจะเป็นดังนี้คือ อ่านอินพุตเข้ามา ดึงเอาซิมโบลิคคีย์ออกมาเพื่อคำนวณค่าของกลุ่ม เอา 1 บวกค่าของกลุ่มแล้วใช้เป็นรีเลตีฟคีย์ของแฟ้มดัชนี สั่งให้อ่านเรคอร์ดจากแฟ้มดัชนีขึ้นมาได้แล้วให้เอาซิมโบลิคคีย์ไล่เปรียบเทียบดูในดัชนีทั้ง 60 ตัว ว่ามีซ้ำกันหรือไม่ ในขณะที่ไล่หาอยู่นั้นก็เช็คไปด้วยว่าถึงดัชนีตัวสุดท้ายหรือยัง (อาจมีลิงค์) เมื่อถึงตัวสุดท้ายและไม่มีคีย์ที่ซ้ำกันก็แสดงว่าเรคอร์ดนี้โหลดได้ ถึงตอนนี้ก็จะต้องไปอ่านเอาเรคอร์ดที่ 0 ของแฟ้มข้อมูลหลักขึ้นมา ทั้งนี้เพื่อดึงค่าของ "Next Available Record" ขึ้นมาใช้งาน เจ้าตัวนี้จะเป็นรีเลตีฟคีย์เพื่อสั่ง WRITE ลงในแฟ้มข้อมูลหลัก เมื่อ WRITE ลงไปแล้วก็มาสร้าง "เอนทรี้" หรือคือจับซิมโบลิคคีย์มาวางติดกับรีเลตีฟคีย์ แล้วนำไปบรรจุลงในเรคอร์ดของแฟ้มดัชนี เสร็จแล้วก็ WRITE กลับลงไว้ในแฟ้มดัชนี เมื่ออัพเดตดัชนีเสร็จแล้วก็ย้อนกลับมาปรับค่า Next Available Record ของในแฟ้มข้อมูลหลัก เมื่อสั่ง SEEK และ WRITE เรคอร์ดที่ 0 กลับลงไปแล้วก็เป็นอันเสร็จการโหลด 1 เรคอร์ด แล้วเราก็วนกลับไปทำต่อจนหมด

ครับ การโหลดข้อมูลลงฐานข้อมูลนี้จะทำกับฐานข้อมูลที่เราเคลียร์เอาไว้แล้วเท่านั้น ดังนั้นฟังก์ชันนี้จึงค่อนข้างจะมีความเสี่ยงสูง ใครไปเรียกใช้โดยไม่ดูให้ดีก็อาจเกิดปัญหาได้ เช่น ไปสั่งโหลดในขณะที่ยังไม่ได้เคลียร์ อย่างนี้จะกลายเป็นการเติมข้อมูลซ้ำเข้าไปในระบบ มันจะถูกดึงออกเพราะมีซิมโบลิคคีย์ที่ซ้ำกับของเดิมเยอะแยะ แต่ในขณะเดียวกัน ถ้าเราจะบอกให้ทำการเคลียร์ฐานข้อมูลเสียก่อนโดยอัตโนมัติ เราก็มีความเสี่ยงจากการถูกใครเรียกใช้ฟังก์ชันนี้โดยไม่ได้ระวัง ฐานข้อมูลอาจสูญหายไปทั้งหมดได้ ดังนั้น ฟังก์ชันโหลด (และอันโหลด) นี้เราควรแยกออกไปเป็นโปรแกรมต่างหาก

ฟังก์ชันอันโหลดข้อมูล

ต่อจากฟังก์ชันโหลดก็จะเป็นเรื่องของการอันโหลด ฟังก์ชันอันโหลดนี้มีประโยชน์ 2 อย่าง อย่างหนึ่งคือ เป็นการสร้างแบคอัพไฟล์ อีกอย่างหนึ่งคือเป็นการทิ้งเรคอร์ดประเภทที่ถูกทำเครื่องหมายถูกลบทิ้งเอาไว้แล้ว วิธีการ



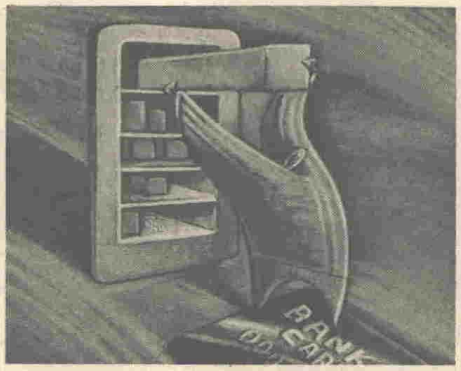
ทำงานของมันคือ เราจะอ่านจากแฟ้มข้อมูลหลักทีละเรคอร์ด เช็คว่าเรคอร์ดนั้นยังมีสถานะ "Active" อยู่หรือไม่ ถ้า "Active" ก็บันทึกออกไปที่ดิสเกตต์เพื่อเป็นแบคอัพ เรคอร์ดประเภท "Deleted" นั้นถ้าท่านอยากจะเก็บไว้เพื่อการอ้างอิงก็อาจบันทึกออกเป็นไฟล์อีกไฟล์หนึ่ง แต่ผมแนะนำให้ทิ้งไปเลย ทั้งนี้เพราะในระบบงานที่ดีนั้นต้องมีรายงานเรื่องการยกเลิกข้อมูลไว้อยู่แล้ว โปรแกรมนี้จะหยุดก็เมื่อไล่ไปจนถึงเรคอร์ดที่มีสถานะ "ว่าง" และเราควรเช็คตำแหน่งของเรคอร์ดนี้ว่าตรงกับค่า Next Available Record ที่บันทึกไว้ในเรคอร์ดที่ 0 (ศูนย์) หรือไม่ด้วย ท่านผู้อ่านควรสังเกตว่าการอันโหลดนี้เราไม่ต้องใช้แฟ้มดัชนีเลย ระบบแบคอัพโดยเอาแต่เฉพาะเรคอร์ดที่ "Active" มาเก็บเป็นซีเควนเชียลไฟล์นี้จะประหยัดเนื้อที่และง่ายต่อการก๊อบปี้ไปใช้งานในเรื่องอื่น เราอาจจะนำเอาไฟล์ที่ได้จากขั้นตอนนี้ไปจัดเรียงตามชื่อหรือเลขประจำตัว หรือตัวเลขอ้างอิง (Reference Number) แล้วพิมพ์เป็นรายงานได้โดยง่าย ผมแนะนำให้ใช้วิธีนี้เป็นตัวแบคอัพข้อมูลในงานทุกชนิด เพราะมันเข้าใจง่าย สามารถส่งข้อมูลไปใช้งานอื่น ๆ ได้ง่ายขึ้น และถ้าจะให้ดียิ่งขึ้นท่านควรเปลี่ยนข้อมูลทุกฟิลด์ให้เป็นชนิดสตริงหรือเท็กซ์เพราะมันสามารถอ่านได้ด้วยโปรแกรมในภาษาอื่น ๆ ได้หมด

เข้าสู่การอัพเดต

โอเค เรารู้วิธีโหลดและอันโหลดข้อมูลแล้วทีนี้ก็จะมาถึงเรื่องของการอัพเดต การอัพเดตนี้ผมแยกไว้เพียง 3 ชนิดใหญ่ คือ "Add Record", "Change Record" และ "Delete Record" เรื่องนี้ผมจะเขียนเป็นโพรซีเดอร์เอาไว้อย่างละโพรซีเดอร์ ซึ่งมีหลักการทำงานดังนี้

ADDRECORD (Symbolic-Key, New-Data, Result) จะเป็นโพรซีเดอร์เพื่อการเติมเรคอร์ดใหม่เข้าไปในไฟล์ เราต้องการ Symbolic-Key เพื่อนำไปเช็คว่ามีซ้ำหรือไม่ ถ้าซ้ำก็จะเซตค่า Result ให้เป็น "1" หรือบอกให้ผู้เรียกใช้ทราบว่ามันเติมไม่ได้เพราะมีคีย์ซ้ำกัน ในกรณีที่ไม่มีคีย์ซ้ำมันก็จะไปอ่านเรคอร์ดที่ 0 (ศูนย์) ของแฟ้มข้อมูลหลักเพื่อหาค่าของ "Next Available Record" มันจะเช็คว่าค่านี้สูงกว่า "End-of-File Pointer" หรือไม่ ถ้าเกินมันจะไม่ทำการอัพเดตให้ มันจะส่งค่า Result เป็น "9" ในกรณีที่มีที่ในแฟ้มข้อมูลหลักมันก็จะไปค้นหาที่สำหรับบรรจุดัชนี นี่ก็เหมือนกัน ถ้าไม่มีที่ที่จะเติมดัชนีมันก็จะไม่อัพเดตให้ และจะส่งค่า Result เป็น "9" แต่ถ้าทุกอย่างปกติดีมันก็จะอัพเดตดัชนี แล้วมาเอา New-data ไปบันทึกลงในแฟ้มข้อมูลหลัก เสร็จแล้วปรับค่า "Next Available Record" ในเรคอร์ดที่ 0 (ศูนย์) เท่านี้ก็เป็นอันเสร็จงาน ในกรณีที่มีการอัพเดตได้ถูกต้อง เราจะเซตค่า Result เป็น "0"

CHANGERECORD (Symbolic-Key, New-Data, Result) จะเป็นโพรซีเดอร์เพื่อใช้แก้ไขข้อมูลที่มีอยู่แล้ว New-Data มีไว้เพื่อการนำไปอัพเดต ซึ่งท่านจะต้องไปแจงรายละเอียดอีกต่อหนึ่ง ท่านอาจเขียนเป็นโพรซีเดอร์ย่อยต่าง ๆ เอาไว้ก็ได้ เมื่อโพรซีเดอร์ CHANGERECORD นี้สามารถค้นหาเรคอร์ดตาม Symbolic Key ที่ให้มาได้ มันก็จะใช้ CASE เพื่อไปดึงโพรซีเดอร์ของท่านมาทำงาน ทำเสร็จก็จะกลับมาเอาเรคอร์ดนั้นบันทึกกลับลงแฟ้มข้อมูลหลักดังเดิม ตัวแปร Result นั้นเราเอาไว้บอกผลการทำงาน คือ "0" หมายถึง ทำงานได้ตามสั่ง "1" หมายถึงมีปัญหาเพราะ Symbolic Key อันนี้ค้นหาแล้วไม่พบ



DELETERECORD (Symbolic-Key, Result) จะเป็นโพรซีเดอร์เพื่อการสั่งยกเลิกการใช้เรคอร์ด โพรซีเดอร์นี้จะใช้ Symbolic-Key เพื่อหาดัชนีในแฟ้มดัชนี ถ้าหาไม่พบก็จะเซตค่า Result เป็น "1" ถ้าหาพบก็จะรู้ว่าเรคอร์ดที่เป็นตัวข้อมูลอยู่ที่ตรงไหน (Relative Key ที่อยู่ในเอนทรี้คือตัวบอกตำแหน่งที่อยู่ของเรคอร์ดในแฟ้มข้อมูลหลัก) โปรแกรมก็จะไปอ่านเอาเรคอร์ดนั้นขึ้นมา แล้วเซตค่าสถานะให้เป็น "2" หลังจากนั้นก็บันทึกกลับลงที่เดิม และแก้ Relative-Key ในดัชนีให้เป็น "0" ถ้าทุกอย่างทำได้เรียบร้อยดีเราก็เซตค่า Result ให้เป็น "0"

รูปที่ 3 และ 4 คือโปรแกรมตัวอย่างที่ใช้สำหรับเคลียร์ไฟล์ อันโหลดไฟล์ โหลดไฟล์ และการอัพเดตทั้ง 3 แบบที่กล่าวมา

ส่งท้าย

ผมเชื่อว่าท่านผู้อ่านสามารถจะทำความเข้าใจได้โดยง่าย ระบบ "Indexed" นี้เป็นระบบที่ง่าย แต่ให้ผลที่ดีมาก คือใช้เวลาเพียง 30-50 มิลลิวินาทีที่จะเข้าถึงข้อมูลที่ต้องการ เวลาที่ใช้ส่วนใหญ่จะอยู่ที่การอ่านเรคอร์ดจากดัชนี 1 ครั้ง และการอ่านเรคอร์ดจากตัวแฟ้มข้อมูลหลักอีก 1 ครั้ง การเติมเรคอร์ด (Add Record) จะเป็นการอัพเดตที่ใช้เวลานานที่สุด คืออ่าน 3 ครั้ง และบันทึก 3 ครั้ง รวมเป็นเวลาประมาณ 120 มิลลิวินาที ท่านผู้อ่านควรสังเกตว่าการอัพเดตไฟล์ที่เขียนไว้นี้จะสั่งเปิดไฟล์และปิดไฟล์อยู่ภายในโพรซีเดอร์แต่ละตัว อันนี้จะทำให้ลดโอกาสที่ไฟล์จะพังหากใครเดินไปเตะปลั๊กไฟหลุด ช่วงเวลานับจากเปิดไฟล์ถึงปิดไฟล์สั้นมาก (คือ 30 ถึง 120 มิลลิวินาทีเท่านั้น) ถ้าใครไปเตะปลั๊กไฟหลุดในขั้นตอนของการอินพุตที่คีย์บอร์ดก็จะไม่มีผลกระทบต่อไฟล์ เราจะไม่ต้องย้อนกลับไปทำงานตั้งแต่เริ่มต้นของวัน

แต่อย่างไรก็ตาม เมื่อเกิดไฟดับโดยกระทันหัน เราควรวิ่งโปรแกรมตรวจเช็คความสอดคล้องของไฟล์เสียก่อน (โปรแกรมนี้ไม่ได้ให้ตัวอย่างมา แต่ท่านคงทำเองได้) ระบบ Indexed Access Method ที่กล่าวมานี้ผมใช้ได้ผลดีมาโดยตลอด จึงขอแนะนำให้ท่านลองนำไปใช้ดู

ต้นฉบับไม่มีหน้า 265-266

No This Page 265-266 In Original

รูปที่ 3 โปรแกรมตัวอย่างที่ใช้สำหรับเคลียร์ไฟล์

```
PROGRAM FT1S01;
LABEL CLEARINDEXFILE,CLEARMASTERFILE,EOJ;
CONST MAXINDX=60;MAXINDXRECORD=120;MAXGROUP=100;
      MAXMASTERRECORD=480;
TYPE  STR5=STRING[5];
      STR30=STRING[30];
      STR45=STRING[45];
TYPE  ENTRYTYPE=RECORD
      SYMBOLICKEY:STR5;
      RELATIVEKEY:INTEGER;
      END;
TYPE  INDEXRECORD=RECORD
      ENTRY:ARRAY[1..MAXINDX] OF ENTRYTYPE;
      FILLER1:STR30;
      LINK:INTEGER;
      END;
TYPE  MASTERRECORD=RECORD
      STATUS:INTEGER;
      ID:STR5;
      DATA:STR45;
      END;
VAR INDEXFILE:FILE OF INDEXRECORD;
VAR INDEXBUFF:INDEXRECORD;
VAR MASTERFILE:FILE OF MASTERRECORD;
VAR MASTERBUFF:MASTERRECORD;
VAR ENTRYBUFF:ENTRYTYPE;
VAR RECNO,K,INDXRECNO:INTEGER;
VAR CH:CHAR;

BEGIN
  CLRSCR;
  GOTOXY(1,1);WRITE('PROGRAM FT1S01 IS RUNNING');
  GOTOXY(1,2);WRITE(' OLD MASTER FILE WILL BE WIPED OUT. ');
  GOTOXY(1,3);WRITE(' ARE YOU SURE? PLEASE CONFIRM (Y/N) ');
  READ(KBD,CH);
  IF CH<>'Y' THEN GOTO EOJ;

CLEARINDEXFILE:
  GOTOXY(1,5);WRITE(' CLEARING THE INDEX-FILE ');
  ASSIGN(INDEXFILE,'FT1D01.DAT');
  REWRITE(INDEXFILE);
  ENTRYBUFF.SYMBOLICKEY:='XXXXX';
  ENTRYBUFF.RELATIVEKEY:=0;
  FOR K:=1 TO MAXINDX DO
    INDEXBUFF.ENTRY[K]:=ENTRYBUFF;
  INDEXBUFF.FILLER1:='                                        ';
  INDEXBUFF.LINK:=0;
  FOR INDXRECNO:=1 TO MAXINDXRECORD DO
    BEGIN
    SEEK(INDEXFILE,INDXRECNO);
    WRITE(INDEXFILE,INDEXBUFF);
    GOTOXY(5,6);WRITE('INDX REC NO ',INDXRECNO:4,' CLEARED');
    END;
  INDEXBUFF.LINK:=MAXGROUP+1;
  SEEK(INDEXFILE,0);
  WRITE(INDEXFILE,INDEXBUFF);
  GOTOXY(5,7);WRITE('INDX REC NO ZERO SET');
  CLOSE(INDEXFILE);

CLEARMASTERFILE:

  GOTOXY(1,8);WRITE(' CLEARING THE MASTER-FILE ');
  ASSIGN(MASTERFILE,'FT1D02.DAT');
  REWRITE(MASTERFILE);
  MASTERBUFF.STATUS:=0;
  MASTERBUFF.ID:='XXXXX';
  MASTERBUFF.DATA:='                                          ';
  FOR RECNO:=1 TO MAXMASTERRECORD DO
    BEGIN
    SEEK(MASTERFILE,RECNO);
    WRITE(MASTERFILE,MASTERBUFF);
    GOTOXY(5,9);WRITE('MASTER REC NO ',RECNO:4,' CLEARED');
    END;
  MASTERBUFF.STATUS:=1;
  SEEK(MASTERFILE,0);
  WRITE(MASTERFILE,MASTERBUFF);
  GOTOXY(5,10);WRITE('MASTER REC ZERO SET');
  CLOSE(MASTERFILE);

EOJ:
  GOTOXY(1,12);WRITE('END OF JOB');
  END.
```

รูปที่ 4 โปรแกรมตัวอย่างที่ใช้สำหรับอันโหลดไฟล์ โหลดไฟล์และการอัพเดต

```
PROGRAM FT1S02;
LABEL MAINLOOP,READKBD,EOJ;
CONST MAXINDX=60;MAXINDXRECORD=120;MAXGROUP=100;
      MAXMASTERRECORD=480;
TYPE  STR5=STRING[5];
      STR45=STRING[45];
TYPE  ENTRYTYPE=RECORD
      SYMBOLICKEY:STR5;
      RELATIVEKEY:INTEGER;
      END;
TYPE  INDEXRECORD=RECORD
      ENTRY:ARRAY[1..MAXINDX] OF ENTRYTYPE;
      FILLER1:STRING[30];
      LINK:INTEGER;
      END;
TYPE  MASTERRECORD=RECORD
      STATUS:INTEGER;
      ID:STR5;
      DATA:STR45;
      END;
VAR CH:CHAR;

PROCEDURE BEEP;
BEGIN
  SOUND(400);DELAY(100);
  NOSOUND;DELAY(50);
END;

PROCEDURE PING;
BEGIN
  SOUND(1100);DELAY(50);
  NOSOUND;DELAY(50);
END;

PROCEDURE NEWSCREEN;
VAR K1:INTEGER;
BEGIN
  CLRSCR;
  FOR K1:=2 TO 79 DO
    BEGIN
    GOTOXY(K1,1);WRITE('*');
    GOTOXY(K1,25);WRITE('*');
    END;
  GOTOXY(35,1);WRITE(' PROGRAM:FT1S02 ');
END;

PROCEDURE CLEARLINE(N:INTEGER);
BEGIN
  GOTOXY(2,N);
  WRITE('                                                  ');
END;

PROCEDURE CALCGROUP(SYMBOLICKEY:STR5;VAR GROUP:INTEGER);
VAR SUM1,SUM2:INTEGER;
VAR CH1,CH2,CH3,CH4,CH5:CHAR;
BEGIN
  CH1:=COPY(SYMBOLICKEY,1,1);
  CH2:=COPY(SYMBOLICKEY,2,1);
  CH3:=COPY(SYMBOLICKEY,3,1);
  CH4:=COPY(SYMBOLICKEY,4,1);
  CH5:=COPY(SYMBOLICKEY,5,1);
  SUM1:=ORD(CH1)+ORD(CH3)+ORD(CH5);
  SUM1:=SUM1 MOD 10;
  SUM2:=ORD(CH2)+ORD(CH4);
  SUM2:=SUM2 MOD 10;
  GROUP:=10*SUM1+SUM2+1;
END;

PROCEDURE ADDRECORD(SYMBOLICKEY:STR5;NEWDATA:STR45;
          VAR RESULT:INTEGER);
LABEL E1;
VAR GROUP,I,DUPLICATE,ENDSEARCH,EMPTYPOSITION:INTEGER;
VAR INDEXRELATIVEKEY,MASTERRELATIVEKEY:INTEGER;
VAR NEXTAVAILABLEINDEX,NEXTAVAILABLEMASTER:INTEGER;
VAR INDEXBUFF,ZEROINDEXBUFF:INDEXRECORD;
VAR MASTERBUFF,ZEROMASTERBUFF:MASTERRECORD;
VAR ENTRYBUFF:ENTRYTYPE;
VAR INDEXFILE:FILE OF INDEXRECORD;
VAR MASTERFILE:FILE OF MASTERRECORD;
BEGIN
  ASSIGN(INDEXFILE,'FT1D01.DAT');
  ASSIGN(MASTERFILE,'FT1D02.DAT');
  RESET(INDEXFILE);
  RESET(MASTERFILE);
  CALCGROUP(SYMBOLICKEY,GROUP);
  INDEXRELATIVEKEY:=GROUP;
  SEEK(INDEXFILE,INDEXRELATIVEKEY);
  READ(INDEXFILE,INDEXBUFF);
  DUPLICATE:=0;ENDSEARCH:=0;EMPTYPOSITION:=0;
```

→

```
  I:=0;
  REPEAT
    I:=I+1;
    IF (INDEXBUFF.ENTRY[I].SYMBOLICKEY='XXXXX') AND
       (INDEXBUFF.ENTRY[I].RELATIVEKEY=0) THEN
       BEGIN ENDSEARCH:=1;EMPTYPOSITION:=I;END;
    IF (I=MAXINDX) AND (INDEXBUFF.LINK=0) THEN
       BEGIN ENDSEARCH:=1;EMPTYPOSITION:=99;END;
    IF (INDEXBUFF.ENTRY[I].SYMBOLICKEY=SYMBOLICKEY) AND
       (INDEXBUFF.ENTRY[I].RELATIVEKEY<>0) THEN DUPLICATE:=1;
    IF (I=MAXINDX) AND (INDEXBUFF.LINK<>0) AND (DUPLICATE=0) THEN
       BEGIN
       INDEXRELATIVEKEY:=INDEXBUFF.LINK;
       SEEK(INDEXFILE,INDEXRELATIVEKEY);
       READ(INDEXFILE,INDEXBUFF);
       I:=0;
       END;
    UNTIL (ENDSEARCH=1) OR (DUPLICATE=1);
  IF DUPLICATE=1 THEN
    BEGIN RESULT:=1;GOTO E1;END;
  SEEK(MASTERFILE,0);
  READ(MASTERFILE,ZEROMASTERBUFF);
  NEXTAVAILABLEMASTER:=ZEROMASTERBUFF.STATUS;
  IF (NEXTAVAILABLEMASTER>MAXMASTERRECORD) THEN
    BEGIN RESULT:=9;GOTO E1;END;
  IF EMPTYPOSITION=99 THEN
    BEGIN
    SEEK(INDEXFILE,0);
    READ(INDEXFILE,ZEROINDEXBUFF);
    NEXTAVAILABLEINDEX:=ZEROINDEXBUFF.LINK;
    IF NEXTAVAILABLEINDEX>MAXINDXRECORD THEN
      BEGIN RESULT:=9;GOTO E1;END;
    INDEXBUFF.LINK:=NEXTAVAILABLEINDEX;
    SEEK(INDEXFILE,INDEXRELATIVEKEY);
    WRITE(INDEXFILE,INDEXBUFF);
    INDEXRELATIVEKEY:=NEXTAVAILABLEINDEX;
    SEEK(INDEXFILE,INDEXRELATIVEKEY);
    READ(INDEXFILE,INDEXBUFF);
    ZEROINDEXBUFF.LINK:=NEXTAVAILABLEINDEX+1;
    SEEK(INDEXFILE,0);
    WRITE(INDEXFILE,ZEROINDEXBUFF);
    END;
  MASTERBUFF.STATUS:=1;
  MASTERBUFF.ID:=SYMBOLICKEY;
  MASTERBUFF.DATA:=NEWDATA;
  MASTERRELATIVEKEY:=NEXTAVAILABLEMASTER;
  SEEK(MASTERFILE,MASTERRELATIVEKEY);
  WRITE(MASTERFILE,MASTERBUFF);
  ZEROMASTERBUFF.STATUS:=NEXTAVAILABLEMASTER+1;
  SEEK(MASTERFILE,0);
  WRITE(MASTERFILE,ZEROMASTERBUFF);
  ENTRYBUFF.SYMBOLICKEY:=SYMBOLICKEY;
  ENTRYBUFF.RELATIVEKEY:=MASTERRELATIVEKEY;
  INDEXBUFF.ENTRY[EMPTYPOSITION]:=ENTRYBUFF;
  SEEK(INDEXFILE,INDEXRELATIVEKEY);
  WRITE(INDEXFILE,INDEXBUFF);
  RESULT:=0;
E1:
  CLOSE(INDEXFILE);
  CLOSE(MASTERFILE);
END;

PROCEDURE CHANGERECORD(SYMBOLICKEY:STR5;NEWDATA:STR45;
          VAR RESULT:INTEGER);
LABEL E1;
VAR GROUP,I,FOUND,ENDSEARCH:INTEGER;
VAR INDEXRELATIVEKEY,MASTERRELATIVEKEY:INTEGER;
VAR INDEXBUFF:INDEXRECORD;
VAR MASTERBUFF:MASTERRECORD;
VAR INDEXFILE:FILE OF INDEXRECORD;
VAR MASTERFILE:FILE OF MASTERRECORD;
BEGIN
  ASSIGN(INDEXFILE,'FT1D01.DAT');
  ASSIGN(MASTERFILE,'FT1D02.DAT');
  RESET(INDEXFILE);
  RESET(MASTERFILE);
  CALCGROUP(SYMBOLICKEY,GROUP);
  INDEXRELATIVEKEY:=GROUP;
  SEEK(INDEXFILE,INDEXRELATIVEKEY);
  READ(INDEXFILE,INDEXBUFF);
  FOUND:=0;ENDSEARCH:=0;
  I:=0;
  REPEAT
    I:=I+1;
    IF (INDEXBUFF.ENTRY[I].SYMBOLICKEY='XXXXX') AND
       (INDEXBUFF.ENTRY[I].RELATIVEKEY=0) THEN ENDSEARCH:=1;
    IF (INDEXBUFF.ENTRY[I].SYMBOLICKEY=SYMBOLICKEY) AND
       (INDEXBUFF.ENTRY[I].RELATIVEKEY<>0) THEN
       BEGIN
       FOUND:=1;
       MASTERRELATIVEKEY:=INDEXBUFF.ENTRY[I].RELATIVEKEY;
       END;
    IF (I=MAXINDX) AND (INDEXBUFF.LINK=0) AND (FOUND=0) THEN
       ENDSEARCH:=1;
    IF (I=MAXINDX) AND (INDEXBUFF.LINK<>0) AND (FOUND=0) THEN
       BEGIN
       INDEXRELATIVEKEY:=INDEXBUFF.LINK;
       SEEK(INDEXFILE,INDEXRELATIVEKEY);
       READ(INDEXFILE,INDEXBUFF);
       I:=0;
       END;
    UNTIL (ENDSEARCH=1) OR (FOUND=1);
  IF FOUND=0 THEN
    BEGIN RESULT:=1;GOTO E1;END;
  SEEK(MASTERFILE,MASTERRELATIVEKEY);
  READ(MASTERFILE,MASTERBUFF);
  MASTERBUFF.DATA:=NEWDATA;
  SEEK(MASTERFILE,MASTERRELATIVEKEY);
  WRITE(MASTERFILE,MASTERBUFF);
  RESULT:=0;
E1:
  CLOSE(INDEXFILE);
  CLOSE(MASTERFILE);
END;

PROCEDURE DELETERECORD(SYMBOLICKEY:STR5;
          VAR RESULT:INTEGER);
LABEL E1;
VAR GROUP,I,FOUND,ENDSEARCH,POSITION:INTEGER;
VAR INDEXRELATIVEKEY,MASTERRELATIVEKEY:INTEGER;
VAR INDEXBUFF:INDEXRECORD;
VAR MASTERBUFF:MASTERRECORD;
VAR INDEXFILE:FILE OF INDEXRECORD;
VAR MASTERFILE:FILE OF MASTERRECORD;
BEGIN
  ASSIGN(INDEXFILE,'FT1D01.DAT');
  ASSIGN(MASTERFILE,'FT1D02.DAT');
  RESET(INDEXFILE);
  RESET(MASTERFILE);
  CALCGROUP(SYMBOLICKEY,GROUP);
  INDEXRELATIVEKEY:=GROUP;
  SEEK(INDEXFILE,INDEXRELATIVEKEY);
  READ(INDEXFILE,INDEXBUFF);
  FOUND:=0;ENDSEARCH:=0;
  I:=0;
  REPEAT
    I:=I+1;
    IF (INDEXBUFF.ENTRY[I].SYMBOLICKEY='XXXXX') AND
       (INDEXBUFF.ENTRY[I].RELATIVEKEY=0) THEN ENDSEARCH:=1;
    IF (INDEXBUFF.ENTRY[I].SYMBOLICKEY=SYMBOLICKEY) AND
       (INDEXBUFF.ENTRY[I].RELATIVEKEY<>0) THEN
       BEGIN
       FOUND:=1;POSITION:=I;
       MASTERRELATIVEKEY:=INDEXBUFF.ENTRY[I].RELATIVEKEY;
       END;
    IF (I=MAXINDX) AND (INDEXBUFF.LINK=0) AND (FOUND=0) THEN
       ENDSEARCH:=1;
    IF (I=MAXINDX) AND (INDEXBUFF.LINK<>0) AND (FOUND=0) THEN
       BEGIN
       INDEXRELATIVEKEY:=INDEXBUFF.LINK;
       SEEK(INDEXFILE,INDEXRELATIVEKEY);
       READ(INDEXFILE,INDEXBUFF);
       I:=0;
       END;
    UNTIL (ENDSEARCH=1) OR (FOUND=1);
  IF FOUND=0 THEN
    BEGIN RESULT:=1;GOTO E1;END;
  SEEK(MASTERFILE,MASTERRELATIVEKEY);
  READ(MASTERFILE,MASTERBUFF);
  MASTERBUFF.STATUS:=2;
  SEEK(MASTERFILE,MASTERRELATIVEKEY);
  WRITE(MASTERFILE,MASTERBUFF);
  INDEXBUFF.ENTRY[POSITION].RELATIVEKEY:=0;
  SEEK(INDEXFILE,INDEXRELATIVEKEY);
  WRITE(INDEXFILE,INDEXBUFF);
  RESULT:=0;
E1:
  CLOSE(INDEXFILE);
  CLOSE(MASTERFILE);
END;

PROCEDURE ACCEPTSYMBOLICKEY(X1,Y1:INTEGER;VAR KEY:STR5);
VAR K:INTEGER;
VAR CH:CHAR;
VAR A:ARRAY[1..5] OF CHAR;
BEGIN
  FOR K:=1 TO 5 DO
    A[K]:=' ';
  K:=0;
```

```
  REPEAT
    READ(KBD,CH);
    IF ((CH>='0') AND (CH<='9')) OR
       ((CH>='A') AND (CH<='Z')) THEN
       BEGIN
       K:=K+1;A[K]:=CH;
       GOTOXY(X1+K-1,Y1);WRITE(CH);
       PING;
       END;
    IF (CH=CHR(8)) THEN
      IF K=0 THEN BEGIN BEEP;BEEP;END ELSE
        BEGIN
        A[K]:=' ';
        GOTOXY(X1+K-1,Y1);WRITE(' ');
        GOTOXY(X1+K-1,Y1);K:=K-1;
        BEEP;
        END;
    UNTIL (K=5);
  KEY:='';
  FOR K:=1 TO 5 DO
    KEY:=KEY+A[K];
END;

PROCEDURE ACCEPTDATA(X1,Y1:INTEGER;VAR DATA:STR45);
VAR K:INTEGER;
VAR CH:CHAR;
VAR A:ARRAY[1..45] OF CHAR;
BEGIN
  FOR K:=1 TO 45 DO
    A[K]:=' ';
  K:=0;
  REPEAT
    READ(KBD,CH);
    IF ((CH>='0') AND (CH<='9')) OR
       ((CH>='A') AND (CH<='Z')) OR (CH=' ') THEN
       BEGIN
       K:=K+1;A[K]:=CH;
       GOTOXY(X1+K-1,Y1);WRITE(CH);
       PING;
       END;
    IF (CH=CHR(8)) THEN
      IF K=0 THEN BEGIN BEEP;BEEP;END ELSE
        BEGIN
        A[K]:=' ';
        GOTOXY(X1+K-1,Y1);WRITE(' ');
        GOTOXY(X1+K-1,Y1);K:=K-1;
        BEEP;
        END;
    UNTIL (K=45) OR (CH=CHR(13));
  DATA:='';
  FOR K:=1 TO 45 DO
    DATA:=DATA+A[K];
END;

PROCEDURE ACCEPTNUM(X1,Y1:INTEGER;VAR NUM:INTEGER);
VAR K:INTEGER;
VAR CH:CHAR;
BEGIN
  NUM:=0;
  K:=0;
  REPEAT
    READ(KBD,CH);
    IF ((CH>='0') AND (CH<='9')) THEN
       BEGIN
       K:=K+1;
       NUM:=10*NUM+(ORD(CH)-48);
       GOTOXY(X1,Y1);WRITE(NUM:4);
       PING;
       END;
    IF (CH=CHR(8)) THEN
      IF K=0 THEN BEGIN BEEP;BEEP;END ELSE
        BEGIN
        NUM:=NUM DIV 10;
        GOTOXY(X1,Y1);WRITE(NUM:4);
        K:=K-1;
        BEEP;
        END;
    UNTIL (K=4) OR (CH=CHR(13));
END;

PROCEDURE DOADDRECORD;
VAR RESULT:INTEGER;
VAR KEY:STR5;
VAR DATA:STR45;
VAR CH:CHAR;
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('ADD NEW RECORD PROCEDURE ');
  GOTOXY(3,10);WRITE('SYMBOLIC KEY=');
  ACCEPTSYMBOLICKEY(16,10,KEY);
  GOTOXY(3,12);WRITE('DATA=');
  ACCEPTDATA(8,12,DATA);
  ADDRECORD(KEY,DATA,RESULT);
  IF RESULT=0 THEN
    BEGIN GOTOXY(3,22);WRITE('RECORD ADDED');END;
  IF RESULT=1 THEN
    BEGIN GOTOXY(3,22);WRITE('DUPLICATE KEY');END;
  IF RESULT=9 THEN
    BEGIN GOTOXY(3,22);WRITE('FILE IS FULL');END;
  READ(KBD,CH);
END;

PROCEDURE DOCHANGERECORD;
VAR RESULT:INTEGER;
VAR KEY:STR5;
VAR DATA:STR45;
VAR CH:CHAR;
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('CHANGE DATA PROCEDURE ');
  GOTOXY(3,10);WRITE('SYMBOLIC KEY=');
  ACCEPTSYMBOLICKEY(16,10,KEY);
  GOTOXY(3,12);WRITE('NEW DATA=');
  ACCEPTDATA(12,12,DATA);
  CHANGERECORD(KEY,DATA,RESULT);
  IF RESULT=0 THEN
    BEGIN GOTOXY(3,22);WRITE('RECORD CHANGED');END;
  IF RESULT=1 THEN
    BEGIN GOTOXY(3,22);WRITE('NOT IN MASTERFILE');END;
  READ(KBD,CH);
END;

PROCEDURE DODELETERECORD;
VAR RESULT:INTEGER;
VAR KEY:STR5;
VAR CH:CHAR;
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('DELETE RECORD PROCEDURE ');
  GOTOXY(3,10);WRITE('SYMBOLIC KEY=');
  ACCEPTSYMBOLICKEY(16,10,KEY);
  DELETERECORD(KEY,RESULT);
  IF RESULT=0 THEN
    BEGIN GOTOXY(3,22);WRITE('RECORD DELETED');END;
  IF RESULT=1 THEN
    BEGIN GOTOXY(3,22);WRITE('NOT IN MASTERFILE');END;
  READ(KBD,CH);
END;

PROCEDURE DODISPLAYRECORD;
VAR RELATIVEKEY:INTEGER;
VAR MASTERBUFF:MASTERRECORD;
VAR MASTERFILE:FILE OF MASTERRECORD;
VAR CH:CHAR;
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('DISPLAY RECORD PROCEDURE ');
  REPEAT
    CLEARLINE(10);
    GOTOXY(3,10);WRITE('RECORD NO=');
    ACCEPTNUM(13,10,RELATIVEKEY);
    UNTIL (RELATIVEKEY>=0) AND (RELATIVEKEY<=MAXMASTERRECORD);
  ASSIGN(MASTERFILE,'FT1D02.DAT');
  RESET(MASTERFILE);
  SEEK(MASTERFILE,RELATIVEKEY);
  READ(MASTERFILE,MASTERBUFF);
  GOTOXY(3,12);WRITE('STATUS=',MASTERBUFF.STATUS);
  GOTOXY(3,14);WRITE('ID    =',MASTERBUFF.ID);
  GOTOXY(3,16);WRITE('DATA  =',MASTERBUFF.DATA);
  CLOSE(MASTERFILE);
  READ(KBD,CH);
END;

PROCEDURE DOBACKUPMASTERFILE;
VAR I,K,LASTRECORD:INTEGER;
VAR MASTERBUFF:MASTERRECORD;
VAR MASTERFILE:FILE OF MASTERRECORD;
VAR BACKUPFILE:FILE OF MASTERRECORD;
VAR CH:CHAR;
```

```
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('BACKUP MASTERFILE PROCEDURE ');
  ASSIGN(MASTERFILE,'FT1D02.DAT');
  RESET(MASTERFILE);
  SEEK(MASTERFILE,0);
  READ(MASTERFILE,MASTERBUFF);
  LASTRECORD:=MASTERBUFF.STATUS-1;
  IF LASTRECORD>0 THEN
    BEGIN
    ASSIGN(BACKUPFILE,'FT1D03.DAT');
    REWRITE(BACKUPFILE);
    I:=0;
    FOR K:=1 TO LASTRECORD DO
      BEGIN
      SEEK(MASTERFILE,K);
      READ(MASTERFILE,MASTERBUFF);
      IF MASTERBUFF.STATUS=1 THEN
        BEGIN
        I:=I+1;
        SEEK(BACKUPFILE,I);
        WRITE(BACKUPFILE,MASTERBUFF);
        GOTOXY(3,10);WRITE('RECORD NO=',K:4,' COPIED');
        END;
      END;
    CLOSE(BACKUPFILE);
    END ELSE
    BEGIN GOTOXY(3,10);WRITE('NO RECORD TO COPY');END;
  CLOSE(MASTERFILE);
  IF LASTRECORD>0 THEN
    BEGIN GOTOXY(3,22);WRITE('BACKUP COMPLETED');END;
  READ(KBD,CH);
END;

PROCEDURE DOLOADMASTERFILE;
VAR K,LASTRECORD,NEXTAVAILABLE,RESULT:INTEGER;
VAR ID:STR5;
VAR DATA:STR45;
VAR MASTERBUFF:MASTERRECORD;
VAR MASTERFILE:FILE OF MASTERRECORD;
VAR BACKUPFILE:FILE OF MASTERRECORD;
VAR CH:CHAR;
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('LOAD MASTERFILE PROCEDURE ');
  ASSIGN(MASTERFILE,'FT1D02.DAT');
  RESET(MASTERFILE);
  SEEK(MASTERFILE,0);
  READ(MASTERFILE,MASTERBUFF);
  NEXTAVAILABLE:=MASTERBUFF.STATUS;
  IF NEXTAVAILABLE=1 THEN
    BEGIN
    CLOSE(MASTERFILE);
    ASSIGN(BACKUPFILE,'FT1D03.DAT');
    RESET(BACKUPFILE);
    LASTRECORD:=FILESIZE(BACKUPFILE)-1;
    FOR K:=1 TO LASTRECORD DO
      BEGIN
      SEEK(BACKUPFILE,K);
      READ(BACKUPFILE,MASTERBUFF);
      ID:=MASTERBUFF.ID;
      DATA:=MASTERBUFF.DATA;
      ADDRECORD(ID,DATA,RESULT);
      GOTOXY(3,10);WRITE('RECORD NO=',K:4,' LOADED');
      END;
    CLOSE(BACKUPFILE);
    END;
  IF NEXTAVAILABLE=1 THEN
    BEGIN GOTOXY(3,22);WRITE('MASTERFILE LOADED');END
    ELSE
    BEGIN GOTOXY(3,22);WRITE('MASTERFILE NOT CLEARED, CANNOT LOAD
  READ(KBD,CH);
END;

BEGIN
  CLRSCR;

MAINLOOP:
  NEWSCREEN;
  GOTOXY(30,5);WRITE('MAIN MENU');
  GOTOXY(30,7);WRITE('1 = ADD NEW RECORD');
  GOTOXY(30,9);WRITE('2 = CHANGE DATA');
  GOTOXY(30,11);WRITE('3 = DELETE RECORD');
  GOTOXY(30,13);WRITE('6 = BACKUP THE MASTERFILE');
  GOTOXY(30,15);WRITE('7 = LOAD THE MASTERFILE');
  GOTOXY(30,17);WRITE('8 = DISPLAY RECORD');
  GOTOXY(30,19);WRITE('9 = END JOB');
  GOTOXY(3,23);WRITE('YOUR SELECTION?');

READKBD:
  GOTOXY(3,24);READ(KBD,CH);
  IF (CH='1') THEN BEGIN DOADDRECORD;GOTO MAINLOOP;END;
  IF (CH='2') THEN BEGIN DOCHANGERECORD;GOTO MAINLOOP;END;
  IF (CH='3') THEN BEGIN DODELETERECORD;GOTO MAINLOOP;END;
  IF (CH='6') THEN BEGIN DOBACKUPMASTERFILE;GOTO MAINLOOP;END;
  IF (CH='7') THEN BEGIN DOLOADMASTERFILE;GOTO MAINLOOP;END;
  IF (CH='8') THEN BEGIN DODISPLAYRECORD;GOTO MAINLOOP;END;
  IF (CH='9') THEN GOTO EOJ;
  BEEP;BEEP;
  GOTO READKBD;

EOJ:
  CLEARLINE(23);CLEARLINE(24);
  GOTOXY(3,23);WRITE('END OF JOB');
  END.
```